# INGI1131 Practical Exercises
# Lab 3: Thread and Declarative Concurrency

This session is dedicated to get your hands dirty with threads and concurrency. Since world is concurrent, better learn how to build concurrent programs. Note also that learning concurrency will help you to write programs that take advantage of multi-core processors.

1. Analyse the following code and, before running it on the OPI, give the values that are bound to variables `X`, `Y` and `Z`, at the end of the execution:

   ```
   local X Y Z in
      thread if X==1 then Y=2 else Z=2 end end
      thread if Y==1 then X=1 else Z=2 end end
      X=1
   end
   ```

   Do the same for the following code:

   ```
   local X Y Z in
      thread if X==1 then Y=2 else Z=2 end end
      thread if Y==1 then X=1 else Z=2 end end
      X=2
   end
   ```

   Verify your answers in the OPI. Of course, you will have to introduce a call to the browser. In which part of the code can you do this?

2. Consider the following procedure:

   ```
   proc {Guess X}
      if X==42 then skip else skip end
   end
   ```

   What does `Guess` do? How does

   ```
   thread {Guess X} <s> end
   ```

   execute? In other words, when is the statement `<s>` executed?

3. Explain what happens when executing the following:

```
declare A B C D in
thread D=C+1 end
thread C=B+1 end
thread A=1 end
thread B=A+1 end
{Browse D}
```

   - In which order are the threads created?
   - In which order are the threads evaluated?

4. The following is a concurrent version of the well known MergeSort algorithm.

```
fun {MergeSort L}
   L1 L2
in
   case L
   of nil then nil
   [] [X] then [X]
   else
      {Split L L1 L2}
      {Merge1 thread {MergeSort L1} end
              thread {MergeSort L2} end}
   end
end
```

   How many threads are created when executing MergeSort with a list of `n` elements. Verify your answer in the OPI for various length lists (you must write your own `Split` and `Merge1` procedures).

5. We will study now the *producer/consumer* concept by doing a simple implementation of it. This topic will be studied more in detail in the following lecture. It is important to notice that the producer and consumer run concurrently.

   A producer is a thread creates a stream of data, for instance integers. A consumer, running in its own thread, will read this stream, and will do something with this data, for instance, compute the sum of all the integers.

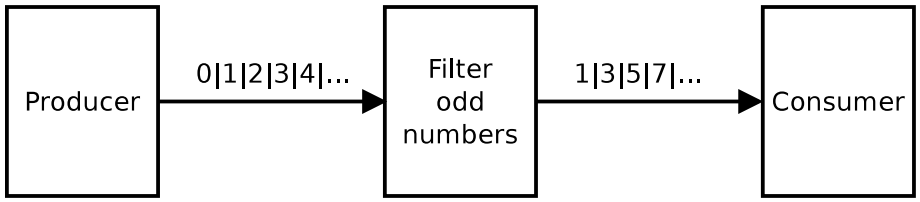
A stream is a list with an unbound tail, as it was shown in with the following code:

```
declare S1 S2 in
S1=a|b|c|d|S2
```

Implement a producer of `N` consecutive integers, starting from 1, and its concurrent consumer that sum up all the elements. How do they synchronize?

6. We can also implements a consumer that produces something, like a filter. Implement a program that asynchronously generates a stream of 10,000 integers, filters out the odd number of the stream and sums up the elements of the resulting stream. The producer, filter, and consumer should all run in separate threads.

   The following figure describes the algorithm, where `Producer` is the process generating the stream of integers and `Consumer` the one summing up the filtered elements.




7. By experiment see approximately how many threads you can run concurrently on your system. How fast can you create this number of threads? Use the Oz Panel. Menu Oz→Open Panel.

   More information here:
   http://www.mozart-oz.org/home/doc/panel/index.html

8. The following code, seen in Lecture 3, synchronizes two procedures to play ping-pong. The synchronization is done using a list. Each procedure blocks on an unbound list and waits that the other procedures binds it to a head and a tail. Once this happens, the procedures browses either `ping` or `pong`, and then binds the rest of the list so that the other process is unblocked.

```
proc {Ping L}
   case L of H|T then T2 in
      {Delay 500} {Browse ping}
      T=_|T2
      {Ping T2}
   end
end

proc {Pong L}
   case L of H|T then T2 in
      {Browse pong}
      T=_|T2
      {Pong T2}
   end
end
```

Consider now the following code call these two procedures:

```
declare L in
thread {Ping L} end
thread {Pong L.2} end
L=_|_
```

- What happens with the execution?
- Why does the game stops?
- How do you fix the program?